Fragment

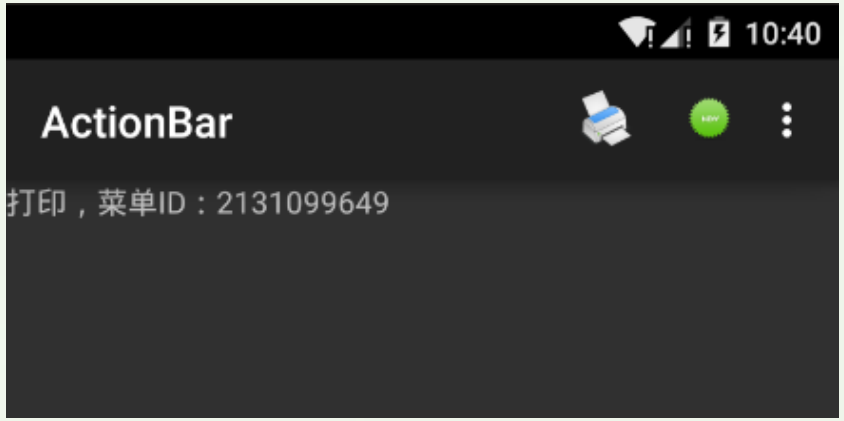
主讲：万永权

4.6 使用Fragment

- 熟悉Fragment的简介和生命周期，能够解释Fragment的定义和生命周期中的方法
- 掌握Fragment的创建方式，能够独立完成Fragment的创建
- 掌握在Activity中添加Fragment的方式，能够独立完成在Activity中添加Fragment

5.5 操作栏与Fragment

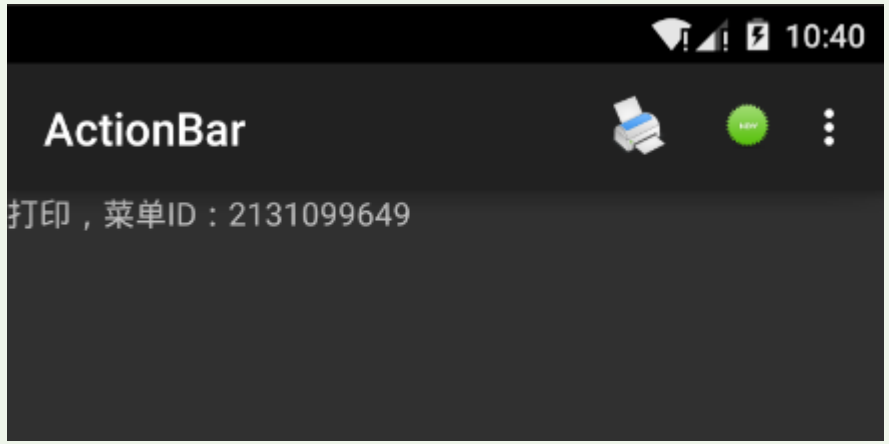
- 操作栏（ActionBar）和Fragment是Android 3.0新引入的界面控件，一定程度上是为了适应Android平板电脑等大屏幕设备界面设计需要而产生的
- 在Android 4.0系统中得到了进一步的发展，可以良好的支持不同屏幕尺寸的设备，并可以根据屏幕大小的不同改变显示内容



5.5 操作栏与Fragment

•5.5.1操作栏

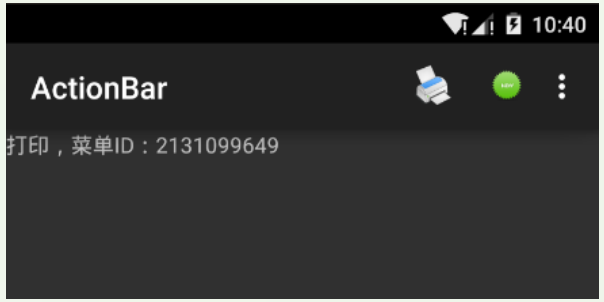
- 操作栏（Action Bar）代替传统的“标题栏”和“选项菜单”功能
- 操作栏左侧的图标是应用程序的图标（Logo），图标旁边是应用程序当前Activity的标题，右侧的多个图标则是“选项菜单”中的菜单项

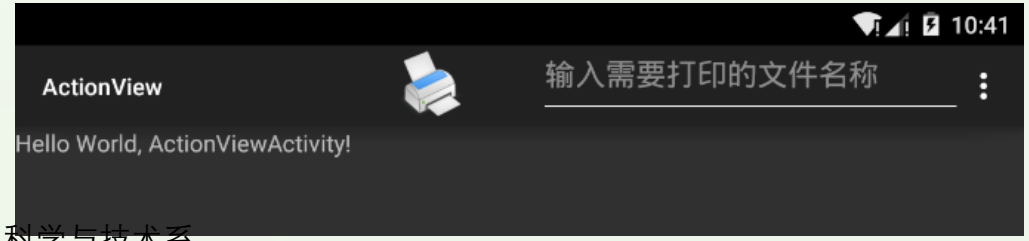


5.5 操作栏与Fragment

•5.5.1操作栏

- 可以提供多个实用的功能：
  - （1）将“选项菜单”的菜单项显示在操作栏的右侧；
  - （2）基于Fragment实现类似于Tab页的导航切换功能；
  - （3）为导航提供可“拖拽—放置”的下拉列表；
  - （4）可在操作栏上实现类似于“搜索框”的功能。

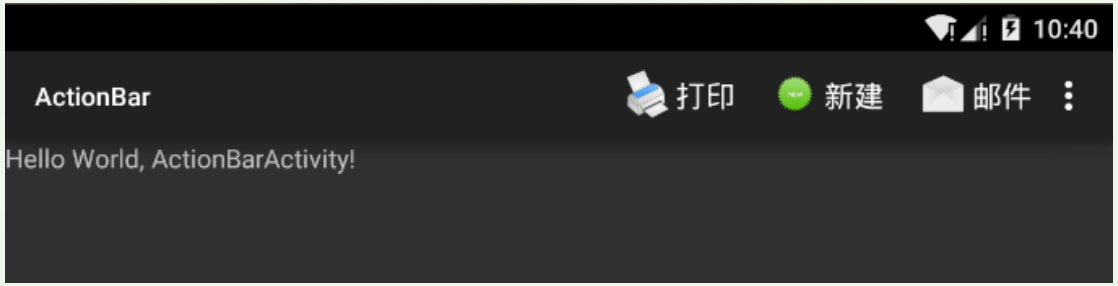


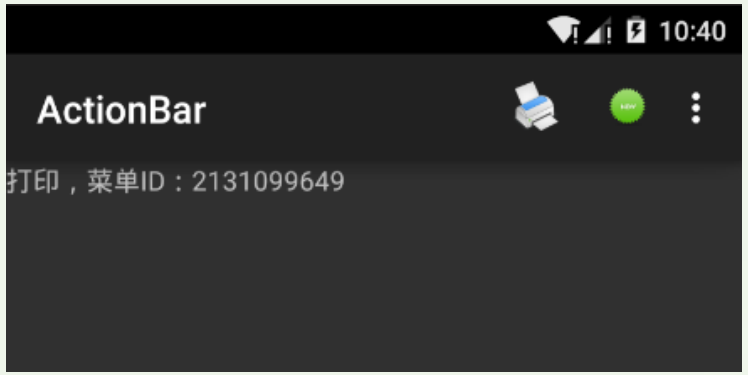


5.5 操作栏与Fragment

•5.5.1操作栏与Fragment

- ActionBar示例
  - 操作栏的实际显示效果，取决于屏幕分辨率和屏幕方向





5.5 操作栏与Fragment

•5.5.1操作栏与Fragment

- ActionBar示例的main_menu.xml文件部分代码：

```
1.  <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.      xmlns:tools="http://schemas.android.com/tools">
3.      <item android:id="@+id/main_menu_0"
4.          android:icon="@drawable/pic0"
5.          android:title="打印"
6.          android:showAsAction="ifRoom|withText"
7.          tools:ignore="AppCompatResource" />
8.  </menu>
```

- 第6行代码中的ifRoom表示如果操作栏有剩余空间，则显示该菜单项的图标；withText表示显示图标的同时显示文字标题

5.5 操作栏与Fragment

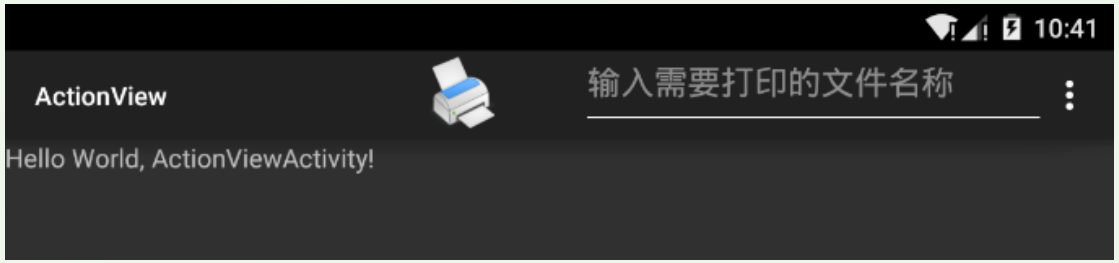
•5.5.1操作栏与Fragment

- ActionBar示例的main_menu.xml文件的完整代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <item android:id="@+id/main_menu_0"
        android:icon="@drawable/pic0"
        android:title="打印"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_1"
        android:icon="@drawable/pic1"
        android:title="新建"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_2"
        android:icon="@drawable/pic2"
        android:title="邮件"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_3"
        android:icon="@drawable/pic3"
        android:title="设置"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
    <item android:id="@+id/main_menu_4"
        android:icon="@drawable/pic4"
        android:title="订阅"
        android:showAsAction="ifRoom|withText"
        tools:ignore="AppCompatResource" />
</menu>
```

5.5 操作栏与Fragment

- **5.5.1操作栏**
  - ActionView示例
  - 在ActionBar示例基础上做的修改
  - 在操作栏上增加了文字输入功能



WXGA720 (1280x720）分辨率下的显示效果

5.5 操作栏与Fragment

•5.5.1操作栏与Fragment

- 在item标签中添加android:actionLayout属性，并将属性值定义为需要显示的布局文件

```
1.  <item android:id="@+id/main_menu_0"
2.          android:icon="@drawable/pic0"
3.          android:title="打印"
4.          android:showAsAction="ifRoom|withText"
5.          tools:ignore="AppCompatResource"
6.          android:actionLayout="@layout/printview" />
```

- 代码第6行表示显示该菜单项时，采用/layout/printview.xml文件作为自定义布局

5.5 操作栏与Fragment

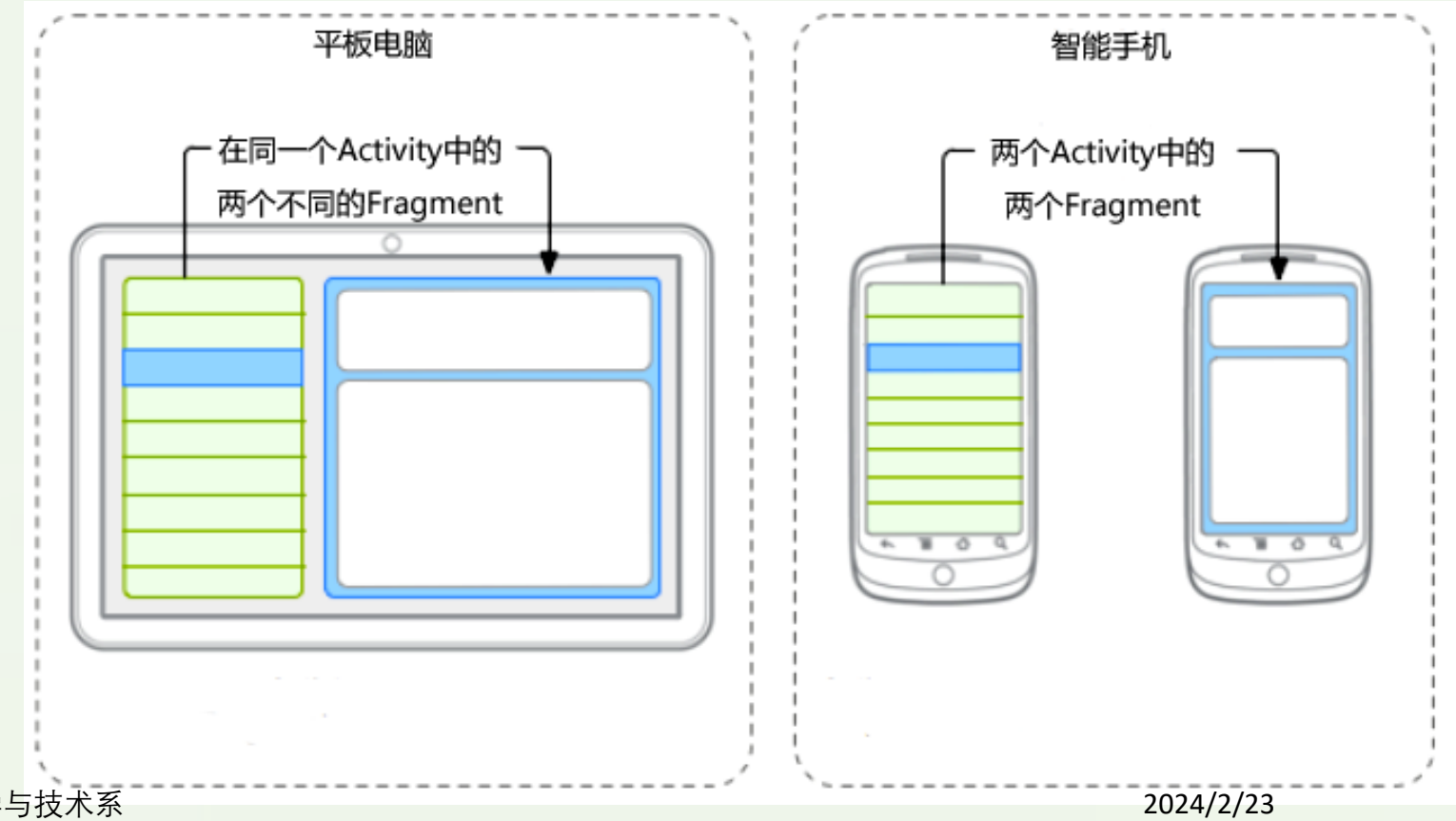
•5.5.1操作栏与Fragment

- printview.xml文件的完整代码如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:orientation="horizontal" >
6.      <ImageView
7.          android:layout_width="wrap_content"
8.          android:layout_height="wrap_content"
9.          android:src="@drawable/pic0" />
10.     <EditText
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:hint="输入需要打印的文件名称"
14.         android:ems="12" />
15. </LinearLayout>
```

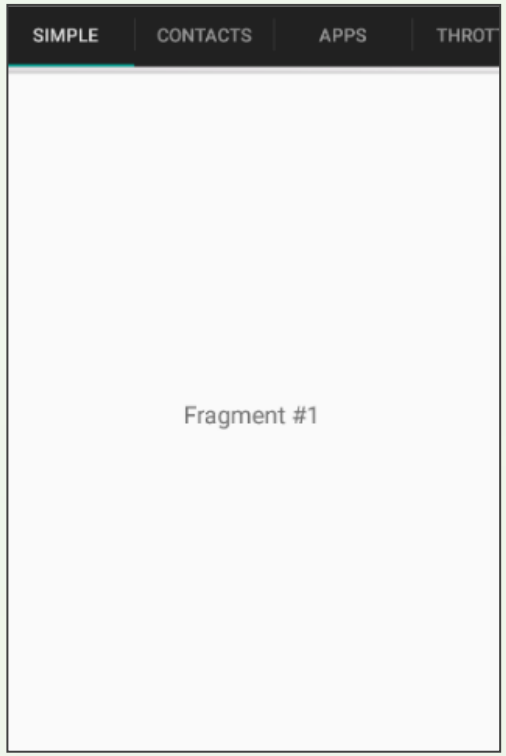
5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - 用途是在大屏幕设备上实现灵活、动态的界面设计



4.6.1 Fragment简介

Fragment（碎片）是一种嵌入在Activity中的UI片段，它可以用来描述Activity中的一部分布局。



Fragment_1

Fragment_2

5.5 操作栏与Fragment

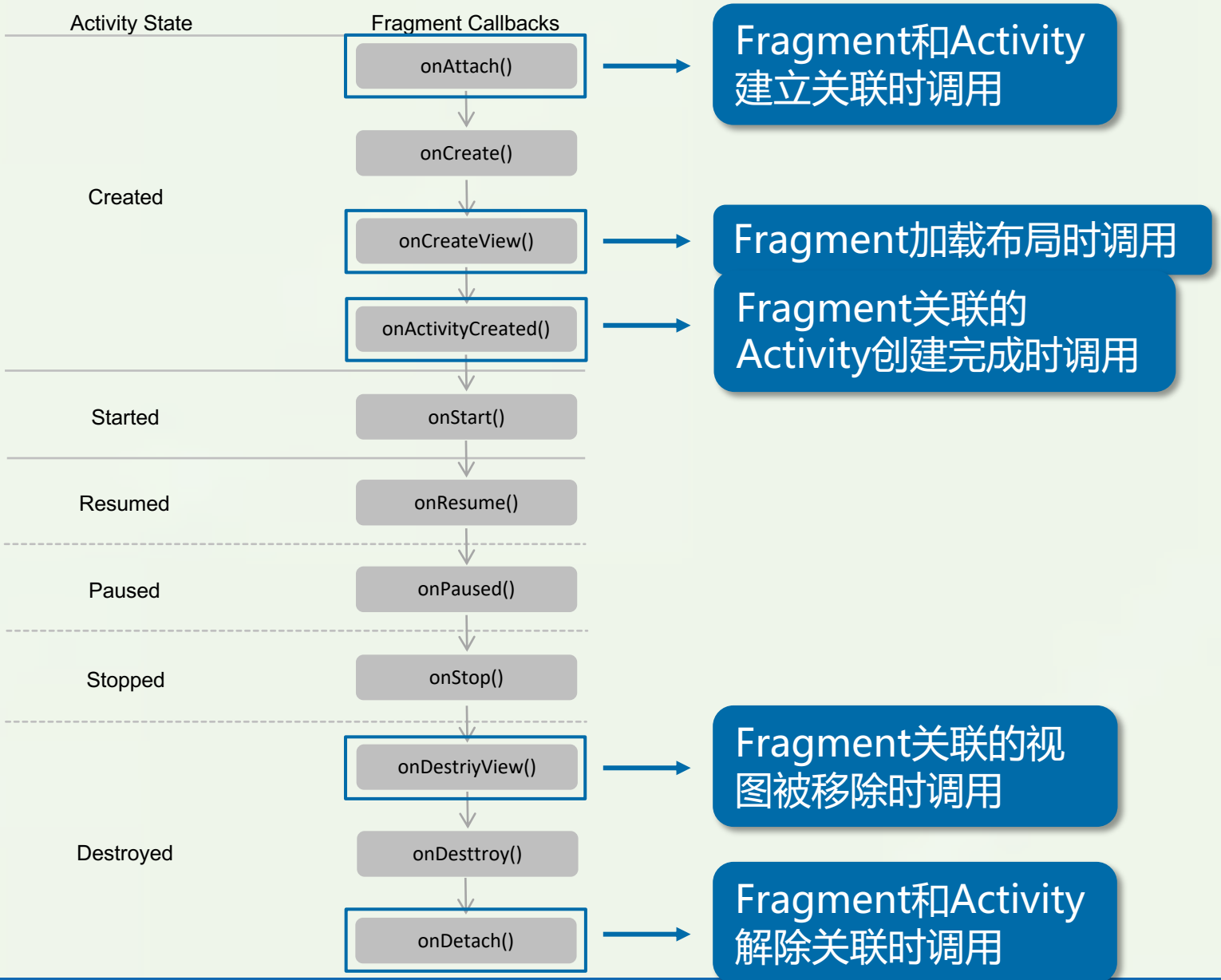
- **5.5.2 Fragment**
  - 可以被设计成为可重用模块的，因为每个Fragment都有自己的布局和生命周期回调函数，可以将同一个Fragment放置到多个不同的Activity中
  - 为了重复使用Fragment，应该避免直接从一个Fragment去操纵另一个Fragment，这样会增加两个Fragment之间的耦合度，不利于模块的重用

4.6.2 Fragment的生命周期

Fragment不能独立存在，必须嵌入到Activity中使用，所以Fragment生命周期直接受所在的Activity影响。

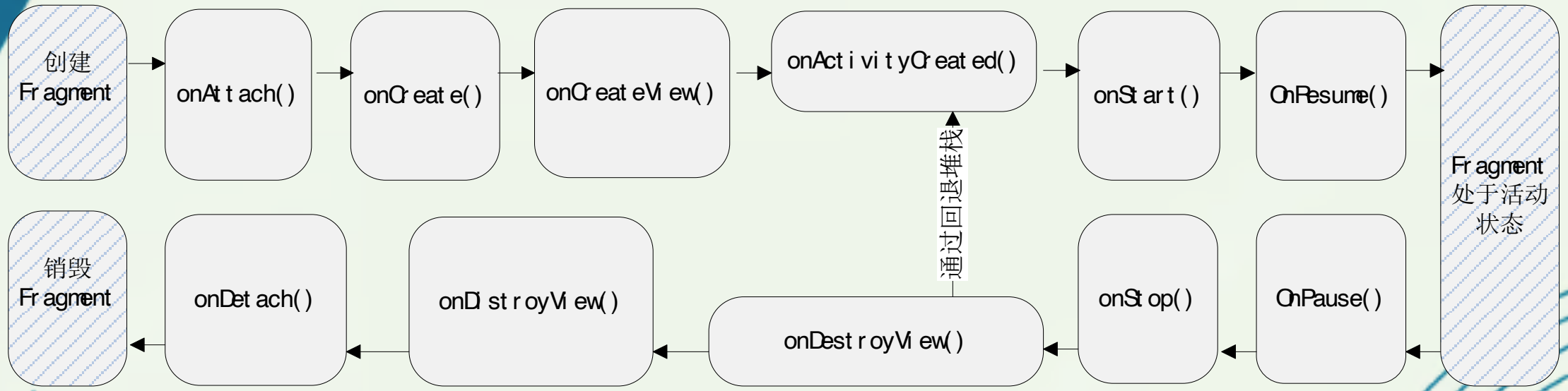
- 当在Activity中创建Fragment时，Fragment处于启动状态；
- 当Activity被暂停时，其中的所有Fragment也被暂停；
- 当Activity被销毁时，所有在该Activity中的Fragment也被销毁。
- 当一个Activity处于运行状态时，可以单独地对每一个Fragment进行操作，如添加或删除，当添加时，Fragment处于启动状态。当删除时，Fragment处于销毁状态。

4.6.2 Fragment的生命周期



5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - Fragment具有与Activity类似的生命周期，但比Activity支持更多的事件回调函数



5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - 通常情况下，创建Fragment需要继承Fragment的基类，并至少应实现onCreate()、onCreateView()和onPause()三个生命周期的回调函数
    - onCreate()函数是在Fragment创建时被调用，用来初始化Fragment中的必要组件
    - onCreateView()函数是Fragment在用户界面上第一次绘制时被调用，并返回Fragment的根布局视图
    - onPause()函数是在用户离开Fragment时被调用，用来保存Fragment中用户输入或修改的内容
  - 如果仅通过Fragment显示元素，而不进行任何的数据保存和界面事件处理，则可仅实现onCreateView()函数

4.6.3 创建Fragment

注 意

Android系统中提供了2个Fragment类，这两个类分别是android.app.Fragment和android.support.v4.app.Fragment。

（1）如果NewsListFragment类继承的是android.app.Fragment类，则程序只能兼容3.0版本以上的Android系统。

（2）如果NewsListFragment类继承的是android.support.v4.app.Fragment类，则程序可以兼容1.6版本以上的Android系统。

4.6.4 在Activity中添加Fragment

Fragment创建完成后并不能单独使用，还需要将Fragment添加到Activity中。

在Activity中添加Fragment有两种方式。

1. 在布局文件中添加Fragment

2. 在Activity中动态加载Fragment

4.6.4 在Activity中添加Fragment

在布局文件中添加Fragment

```
<fragment
    android:name="cn.itcast.NewsListFragment"
    android:id="@+id/newslist"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Fragment的全路径名称

4.6.4 在Activity中添加Fragment

在Activity中动态加载Fragment

当Activity运行时，也可以将Fragment动态添加到Activity中，具体步骤如下：

（1） 创建一个Fragment的实例对象。

（2） 获取FragmentManager类的实例。

（3） 开启FragmentTransaction(事务)。

（4） 向Activity的布局容器(一般为FrameLayout)中添加Fragment。

（5） 通过commit()方法提交事务。

4.6.4 在Activity中添加Fragment

在Activity中动态加载Fragment

```
NewsListFragment fragment = new NewsListFragment
FragmentManager fm = getFragmentManager();
FragmentTransaction beginTransaction = fm.beginTransaction();
beginTransaction.replace(R.id.ll,fragment);
beginTransaction.commit();
```

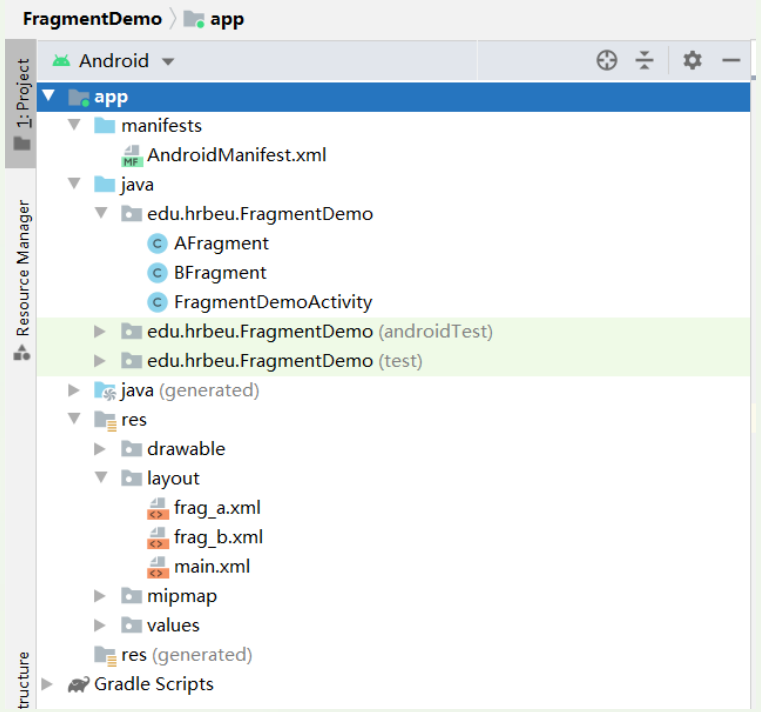
开启事务

添加一个Fragment

提交事务

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - 说明如何在一个Activity中同时加载两个Fragment



5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - main.xml文件是Activity的布局文件，两个Fragment在界面上的位置关系就在这个文件中进行的定义

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name ="edu.hrbeu.FragmentDemo.AFragment"
          android:id="@+id/fragment_a"
          android:layout_weight="1"
          android:layout_width="0px"
          android:layout_height="match_parent" />
     <fragment android:name ="edu.hrbeu.FragmentDemo.BFragment"
          android:id="@+id/fragment_b"
          android:layout_weight="1"
          android:layout_width="0px"
          android:layout_height="match_parent" />
</LinearLayout>
```

5.5 操作栏与Fragment

•5.5.2 Fragment

- FragmentDemo示例
  - FragmentDemoActivity是该示例主界面的Activity，加载了main.xml文件声明的界面布局
  - FragmentDemoActivity.java文件的完整代码如下:

```
1.  public class FragmentDemoActivity extends Activity {
2.      @Override
3.      public void onCreate(Bundle savedInstanceState) {
4.          super.onCreate(savedInstanceState);
5.          setContentView(R.layout.main);
6.      }
7.  }
```

  - Android系统会根据代码第5行的内容加载界面布局文件main.xml，然后通过main.xml文件中对Fragment所在的“包+类”的描述，找到Fragment的实现类，并调用类中的onCreateView()函数绘制界面元素

5.5 操作栏与Fragment

•5.5.2 Fragment

- FragmentDemo示例
  - AFragment.java文件的核心代码如下:

```
1.  public class AFragment extends Fragment{
2.  @Override
3.      public View onCreateView(LayoutInflater inflater, ViewGroup container,
4.                               Bundle savedInstanceState) {
5.          return inflater.inflate(R.layout.frag_a, container, false);
6.      }
7.  }
```

  - AFragment中只实现了onCreateView()函数(代码第3行)，返回值是AFragment的视图
  - 代码第5行使用inflate()函数，通过指定资源文件R.layout.frag_a，获取到AFragment的视图

5.5 操作栏与Fragment

- **5.5.2 Fragment**
  - FragmentDemo示例
    - 最后给出frag_a.xml文件的全部代码如下:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:orientation="vertical" >
6.      <TextView
7.          android:layout_width="wrap_content"
8.          android:layout_height="wrap_content"
9.          android:text="AFragment" />
10.     <TextView
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:text="这是AFragment的显示区域，通过这行文字可以看到与BFragment的边界" />
14.     <CheckBox
15.         android:layout_width="wrap_content"
16.         android:layout_height="wrap_content"
17.         android:text="AF选项" />
18.     <Button
19.         android:layout_width="wrap_content"
20.         android:layout_height="wrap_content"
21.         android:text="AF按钮" />
22. </LinearLayout>
```

4.6.5 实战演练—仿美团菜单

本节我们以仿美团外卖菜单的案例为例来演示如何在一个Activity中展示两个Fragment，并实现Activity与Fragment之间的通信功能。本案例的界面效果如下图所示。

1 搭建菜单界面布局：

①搭建左侧菜单栏界面布局
②搭建右侧菜单列表界面布局
③搭建菜单列表界面的条目布局

2 实现菜单界面功能：

①封装菜品信息的实体类
②加载左侧菜单栏界面布局
③编写菜单列表的适配器
④加载右侧菜单栏界面布局
⑤实现显示菜单的效果

4.6.5 实战演练—仿美团菜单

3 **运行程序：** ①运行程序，显示推荐选项的界面效果
②点击“进店必买选项”，显示其对应的界面效果

5.5 操作栏与Fragment

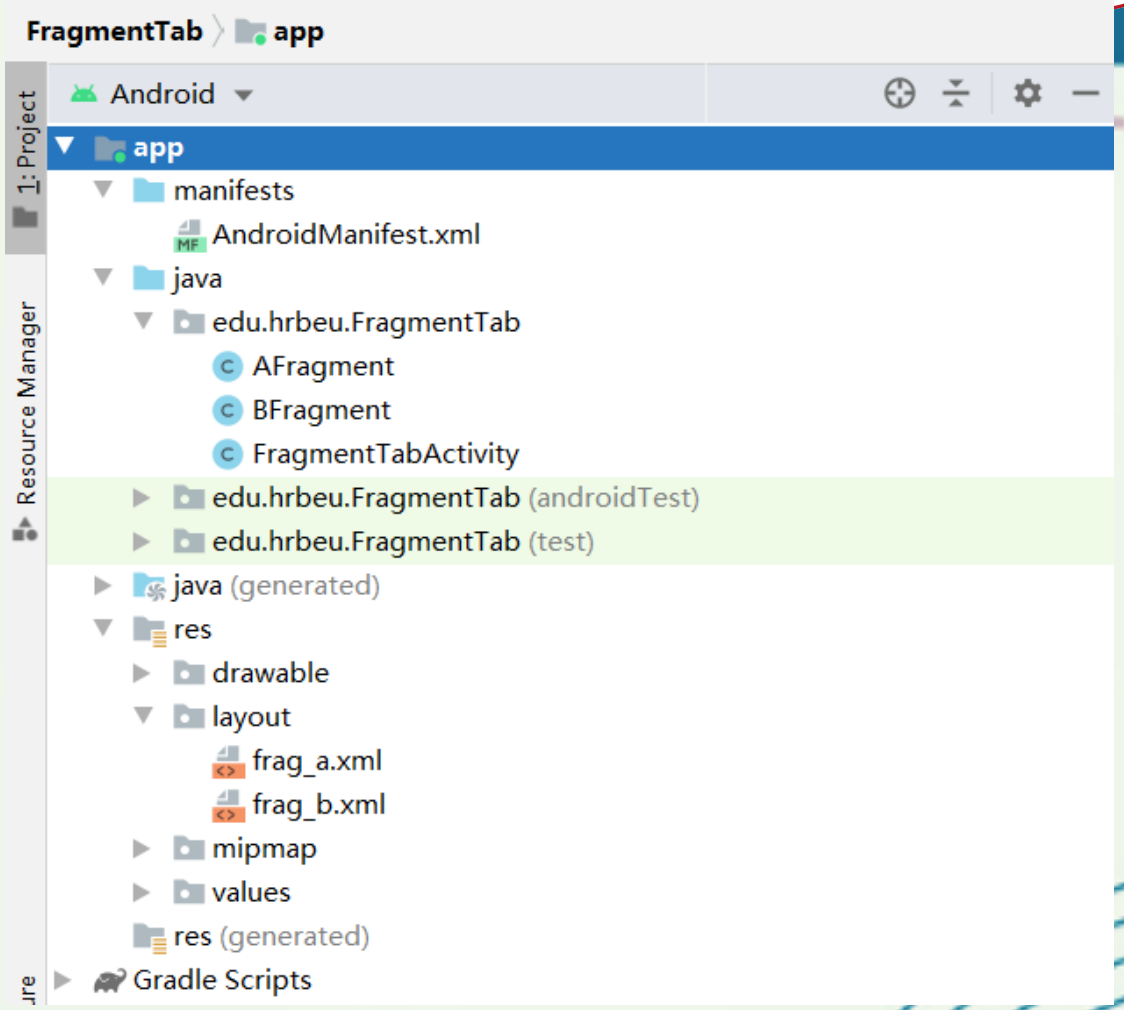
•5.5.2 Tab导航栏

- 在界面控件的章节中介绍过使用TabHost和TabActivity实现Tab导航栏的功能，但因为TabActivity已经过期，所以这里介绍一种新方法，使用操作栏和Fragment实现Tab导航栏。
- 下面用FragmentTab示例说明如何使用操作栏和Fragment实现Tab导航栏，FragmentTab示例的用户界面如图:

5.5 操作栏与Fragment

•5.5.2 Tab导航栏

- Tab导航栏介绍
  - 第一个Tab页的标题为 “FRAGMENT A”，第二个Tab页的标题为 “FRAGMENT B”，两个Tab页分别加载了不同Fragment，两个Fragment所显示的界面元素略有不同。从图5.35的文件结构可以看得出来，FragmentTab示例和FragmentDemo示例中的一部分文件的文件名称是完全相同的，这些文件中的代码也是完全相同的。这些文件包括AFragment.java、BFragment.java、frag_a.xml和frag_b.xml。这里就不再给出上述文件的源代码，读者可以参考FragmentDemo示例。



5.5 操作栏与Fragment

•5.5.2 Tab导航栏

- Tab导航栏介绍
  - 建立Tab导航栏代码，以及将导航栏和Fragment关联起来的代码都在FragmentTabActivity.java文件中。下面分别介绍FragmentTabActivity.java文件中的核心函数。
  - 先给出onCreate()函数的代码：

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        final ActionBar bar = getActionBar();
        bar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        bar.setDisplayOptions(0, ActionBar.DISPLAY_SHOW_TITLE);
        bar.addTab(bar.newTab()
                .setText("Fragment A")
                .setTabListener(new TabListener<AFragment>(
                        this, "fa",AFragment.class)));
        bar.addTab(bar.newTab()
                .setText("Fragment B")
                .setTabListener(new TabListener<BFragment>(
                  this, "fb", BFragment.class)));
        if (savedInstanceState != null) {
            bar.setSelectedNavigationItem(savedInstanceState.getInt("tab", 0));
        }
    }
```

5.5 操作栏与Fragment

•5.5.2 Tab导航栏

- Tab导航栏介绍
  - 代码第5行调用getActionBar()获取操作栏实例。
  - 代码第6行将操作栏的导航模式设置为Tab导航栏模式，NAVIGATION_MODE_TABS常量的值为2。还支持的常量包括NAVIGATION_MODE_LIST（值为1）和NAVIGATION_MODE_STANDARD（值为0），分别表示列表导航栏和标准导航栏。
  - 代码第7行用来设置操作栏的显示选项。setDisplayOptions(int options, int mask)函数的options参数表示显示的内容，而mask参数则表示不显示的内容。第7行代码的意思是关闭“显示标题文字(DISPLAY_SHOW_TITLE)”。setDisplayOptions()函数支持的常量如表5.4所示。
  - 代码第9行使用add()函数添加Tab页，代码第10行设置Tab页的标题，代码第11行定义这是Tab页点击事件的监听函数。
  - 代码第16行和第17行，表明如果Activity不是首次启动，则在savedInstanceState变量中获取当前Tab页的索引号。

5.5 操作栏与Fragment

5.5.2 Tab导航栏

Tab导航栏介绍

setDisplayOptions()函数支持的常量

常量	值	说明
DISPLAY_HOME_AS_UP	4	在Home元素左侧显示回退按钮
DISPLAY_SHOW_CUSTOM	16	显示自定义视图
DISPLAY_SHOW_HOME	2	在操作栏中显示Home元素
DISPLAY_SHOW_TITLE	8	显示Activity的标题
DISPLAY_USE_LOGO	1	使用Logo代替程序图标

5.5 操作栏与Fragment

•5.5.2 Tab导航栏

- Tab导航栏介绍
  - onSaveInstanceState()函数在Activity临时推出时，将当前Tab页的索引号保存在Bundle中，代码如下：

```
1.      @Override
2.       protected void onSaveInstanceState(Bundle outState) {
3.          super.onSaveInstanceState(outState);
4.          outState.putInt("tab", getActionBar().getSelectedNavigationIndex());
5.       }
```

  - 构造Tab导航栏的事件监听函数，必须实现ActionBar.TabListener接口，主要是实现接口中3个函数，分别是onTabSelected()、onTabUnselected()和onTabReselected()。onTabSelected()在当前Tab页被选中时调用，onTabUnselected()在其它Tab页被选中时调用，onTabReselected()在当前Tab页被再次选中时调用。

5.5 操作栏与Fragment

•5.5.2 Tab导航栏

- Tab导航栏介绍
  - 静态类TabListener的代码如下：

```
1    public static class TabListener<T extends Fragment> implements ActionBar.TabListener {
2        private final Activity mActivity;
3        private final String mTag;
4        private final Class<T> mClass;
5        private final Bundle mArgs;
6        private Fragment mFragment;
7
8        public TabListener(Activity activity, String tag, Class<T> clz) {
9            this(activity, tag, clz, null);
10       }
11
12       public TabListener(Activity activity, String tag, Class<T> clz, Bundle args) {
13           mActivity = activity;
14           mTag = tag;
15           mClass = clz;
16           mArgs = args;
17
18           mFragment = mActivity.getFragmentManager().findFragmentByTag(mTag);
19           if (mFragment != null && !mFragment.isDetached()) {
```

5.5 操作栏与Fragment

5.5.2 Tab导航栏

Tab导航栏介绍

静态类TabListener的代码如下：

```
20              FragmentTransaction ft = mActivity.getFragmentManager().beginTransaction();
21              ft.detach(mFragment);
22              ft.commit();
23          }
24      }
25
26      public void onTabSelected(Tab tab, FragmentTransaction ft) {
27          if (mFragment == null) {
28              mFragment = Fragment.instantiate(mActivity, mClass.getName(), mArgs);
29              ft.add(android.R.id.content, mFragment, mTag);
30          } else {
31              ft.attach(mFragment);
32          }
33      }
34
35      public void onTabUnselected(Tab tab, FragmentTransaction ft) {
36          if (mFragment != null) {
37              ft.detach(mFragment);
38          }
39      }
40
41      public void onTabReselected(Tab tab, FragmentTransaction ft) {
42          Toast.makeText(mActivity, "Reselected!", Toast.LENGTH_SHORT).show();
43      }
44  }
```

5.5 操作栏与Fragment

- **5.5.2 Tab导航栏**
  - Tab导航栏介绍
    - 静态类TabListener的代码
      - FragmentTransaction是封装了Fragment变换所要用的函数，包括将Fragment加入到Activity的add()函数，将Fragment从当前界面分离的Detach()函数，将被Detach()函数分离的Fragment重新连接到界面的attach()函数
      - 上面的代码具有一定的难度，部分内容涉及到Java泛型编程的内容，例如代码第1行和第12行，读者可以参考Java语言的相关资料